Peersum : Gestion des résumés de données dans les systèmes P2P

Rabab Hayek, Guillaume Raschia, Patrick Valduriez, Noureddine Mouaddib

PeerSum: Summary Management in P2P Systems

Rabab Hayek†, Guillaume Raschia†, Patrick Valduriez‡ and Noureddine Mouaddib†

Atlas team–INRIA and LINA, University of Nantes
2 rue de la Houssiniere BP 92208, 44 322 Nantes, FRANCE

†{surname}.{name}@univ-nantes.fr, ‡Patrick.Valduriez@inria.fr

Abstract

Sharing huge, massively distributed databases in P2P systems is inherently difficult. As the amount of stored data increases, data localization techniques become no longer sufficient. A practical approach is to rely on compact database summaries rather than raw database records, whose access is costly in large P2P systems.

In this paper, we consider summaries that are synthetic, multidimensional views with two main virtues. First, they can be directly queried and used to approximately answer a query without exploring the original data. Second, as semantic indexes, they support locating relevant nodes based on data content. The main contribution of this paper is to define an efficient algorithm for partitioning an unstructured P2P network into domains, in order to optimally distribute summaries in the network. Then, we propose a distributed algorithm for maintaining a summary in a given domain. Our performance evaluation shows that the cost of query routing is minimized, while incurring a low cost of summary maintenance.

Keywords: P2P systems, DB summarization

1 Introduction

Research on P2P systems is focusing on supporting advanced applications which must deal with semantically rich data (e.g. XML documents, relational tables, etc.) using a high-level SQL-like query language. As a potential example of applications, consider the cooperation of scientists who are willing to share their private data for the duration of a given experiment. Such cooperation may be efficiently supported by improving the mechanisms of data localization and data description.

In unstructured P2P systems, query routing relies on flooding mechanisms which suffer from high query execution cost and poor recall. To improve performance, several techniques have been proposed to locate data relevant to a user query. These techniques can be grouped in three classes: data indexing, mediation and content-based clustering. Data indexing maintains the location (e.g. [1], [2]) or the direction (e.g. [3]) to nodes storing relevant data. However, efficient data indexes must be small, distributed and refer to data based on their content, without compromising peer autonomy or mandating a specific network structure. Mediation consists in exploiting structural information on database schemas to guide query propagation. For instance, in Piazza [4], a query is propagated along pre-existing pairwise mappings between peer schemas. However, many limitations prevent these techniques from scaling up. Content-based clustering consists in organizing the network such that "similar" peers, e.g. peers answering similar queries, are grouped together ([5], [6]). Similarity between peers may be computed using techniques of the two preceding classes (e.g. similarity between indexes [7]).

With the ever increasing amount of information stored into databases, data localization techniques are no longer sufficient to support P2P data sharing. Today's decision-support and collaborative applications are typically exploratory. Thus, a user may prefer

a fast, approximate answer to a long, exact answer. In other words, reasoning on compact data descriptions rather than raw database records, whose access is costly in large P2P systems, may be much more efficient. For instance, a doctor asking queries like "*young* and *fat* male patients diagnosed with disease X" may prefer descriptions of result tuples to rapidly make a decision based on similar situations, treated by other doctors.

Our work aims at managing summaries over shared data in P2P systems. Data summaries are synthetic, multidimensional views with two main virtues. First, they provide an intelligible representation of the underlying data such that an approximate query can be processed entirely in their domain; that is, inputs and outputs are summaries. Second, as indexing structures, they support locating relevant nodes based on their data descriptions.

This paper makes the following contributions. First, we define an efficient algorithm for partitioning an unstructured P2P network into domains, in order to optimally distribute summaries in the network. Then, we propose a distributed algorithm for managing a summary in a given domain. We validated our algorithmic solutions through simulation, using the BRITE topology generator and SimJava. The performance results show that the cost of query routing is minimized, while incurring a low cost of summary maintenance.

The rest of this paper is organized as follows. Section 2 describes our summary model for P2P systems. Section 3 presents the algorithm for network organization, while section 4 presents the algorithm for summary management. Section 5 discusses query processing in the context of summaries. Section 6 gives a performance evaluation with a cost model and a simulation model. Section 7 compares our solution with related work. Section 8 concludes.

2 Summary Model for P2P Systems

In this section, we first present our summary model architecture and the principle of summary construction in P2P systems. Second, we describe the summarization process that is integrated to a peer DataBase Management System (DBMS), to allow generating summaries of a relational database. Then, we formally define the notion of data summary in a P2P network.

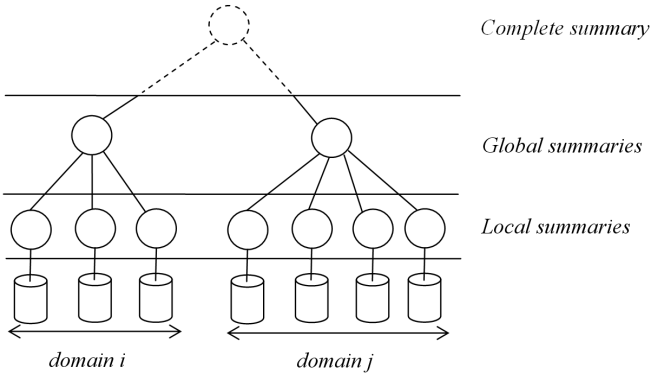

Figure 1. Summary Model Architecture

2.1 Model Architecture

As for any indexing structure in P2P systems, the main issue is how to build and maintain summaries in the network. The data indexing techniques that have been proposed so far fall in two main categories: *local* and *global* indexing. In Gnutella-style systems [8], each node maintains a local index over the data it owns, and query routing relies on flooding mechanisms. Though simple and robust, this approach suffers from high query execution cost and poor recall. A second approach consists in constructing a global index over the shared data, which can be either centralized or distributed in the network. Napster [9] provides search facilities by using a central server that contains an index of all the files every node is sharing. Such a centralized search is very efficient since a single message allows resolving the query. However, a central index is vulnerable to attack and it is difficult to keep it up-to-date.

A distributed-index approach, which we adopt in this paper, maintains indexes at each node. Structured systems (e.g. Pastry [10], Chord [1] and CAN [2]) build a distributed hash table (DHT) on the top of the overlay to organize data in the network. These systems offer an efficient search, but they compromise peer autonomy. Data placement and overlay topology are both tightly controlled. Furthermore, data is located using unique and globally known data identifiers. Thus complex queries are hard to support. Therefore, we leverage the idea of improving the scalability of unstructured P2P systems, rather than turning to DHT-based systems.

Our approach consists in organizing the network into *domains*. Each node maintains a local summary over its own data, and the nodes which are in one domain build a merged (global) summary over the data they share in the domain (Figure 1). The set of global materialized summaries and some links between nodes of their domains, provide a virtual *complete* summary, which ideally describes the content of all data shared in the network. Unlike [11], we do not suppose that nodes are assigned to domains at random. However, we propose a method that partitions the network into d domains, without mandating a specific structure or restricting peer autonomy. As recommended in [12], our method accounts for node heterogeneity, relying on self-inspection to exploit the differences in the nodes' characteristics (e.g. node connectivity). This method is presented in Section 3. However, we will first give a brief description of the summarization process that generates summaries of relational databases with interesting features, making it scalable in a distributed environment.

2.2 Summarization Process

A summarization process is integrated to each peer's DBMS to allow constructing the local summary level of Figure 1. Our approach is based on SAINTETIQ [13], an online linguistic approach for summarizing databases. The summarization process performs a semantic compression of a relational database, that is, it deals with intentional characterization of groups of tuples, using linguistic variables [14]. This semantic compression respects the original dataset schemas, and can directly be queried or used as an alternative dataset for any operation that requires a reduced view of the database (e.g. querying, data mining, browsing, etc.).

A service-oriented architecture of the summary system has been designed thanks to autonomous agents that interact through one-way messages. Figure 2 shows the overall organization of the system into two separate web services. The *translation service* corresponds to the pre-processing step that prepares data for summarization while the *summarization service* produces summaries [15]. In the following we describe these two services. Then, we study the scalability of our summarization process.

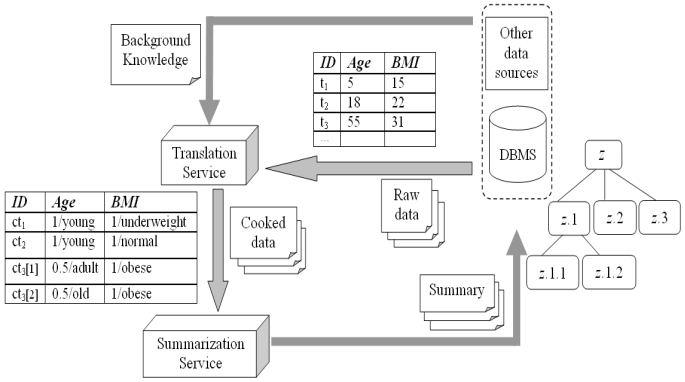

Figure 2. Service-Oriented Architecture of the Summarization Process

2.2.1 Translation Service

A unique feature of the summary system is its extensive use of *Background Knowledge* (BK), a priori built on each attribute. It supports the translation of descriptions of database tuples into a user-defined vocabulary. To best reflect the way users manipulate knowledge about their data, the summary system relies on Zadeh's fuzzy set theory [16] and, more specifically on linguistic variables [14] and fuzzy partitions [17] in order to grasp the inherent imprecision and vagueness of natural language. Descriptors used for summary content representation are defined as linguistic variables on the attribute domain. For example, Figure 3 shows a user-defined vocabulary on the attribute `age` of a `patient` relation[1], where descriptor `adult` is defined as being plainly satisfactory to describe values between 23 and 50 and less satisfactory as the age is out of this range. As shown by Figure 2, the translation service takes as input *raw data* and transforms it to *cooked data*. It supports the process in finding the best representation of an original database tuple, according to the BK provided by the user. Basically, the operation consists in replacing the original value of all the attributes by the set of descriptors defined in the BK that have a non-zero matching value. The flexibility in the vocabulary definition of the BK permits to express any single value with more than one descriptor; in this case, a tuple can be rewritten into several

[1]Body Mass Index (BMI) attribute is defined as the patient's body weight divided by the square of the height.

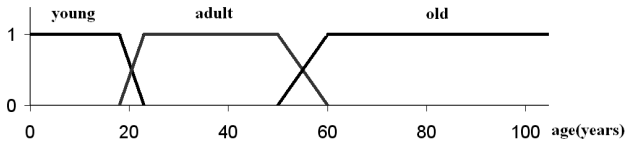

Figure 3. Fuzzy Linguistic Partition on `age`

candidate tuples, as it is exemplified in the Figure 2. Cooked data are documents that contain a single or a collection of candidate tuples and are the input of the summarization service.

2.2.2 Summarization Service

The summarization service performs a clustering task. It takes cooked data as input, and outputs a collection of summaries hierarchically arranged, and incrementally maintained from the root (the most generic summary) to the leaves (the most specific ones). Within this structure, any non-leaf summary generalizes the content of its children nodes. New data, prepared in the form of candidate tuples, are first incorporated in the root node of the hierarchy one at a time. Then, in a top-down conceptual clustering approach based on Fisher's Cobweb algorithm [18], data are processed from the root to the leaves. At each node z, the algorithm considers *incorporating* the current candidate tuple ct into each child node of z as well as *creating* a new child node accommodating ct. Furthermore, the system evaluates the preference of *merging* the two best children nodes of z and *splitting* the best child node. Then, we use a heuristic objective function to determine the best operator to apply at each level of the hierarchy. More details are available in [13], [15].

2.2.3 Scalability Issues

Memory consumption and time complexity are the two main factors that need to be taken care off in order to guaranty the capacity of the summary system to handle massive datasets. First, the time complexity of the SAINTETIQ process is in $O(n)$, where n is the number of candidate tuples to incorporate into a hierarchy of summaries. However, the number of candidate tuples that are produced by the translation service is dependent only on the fuzziness of the BK definition. A crisp BK will produce exactly as many candidate tuples as there are original tuples. Besides, an important feature is that in the summary algorithm, raw data have to be parsed only once, and this is performed with a low time cost. Second, the system requires low memory consumption for performing the summary construction algorithm as well as for storing the produced summaries. Moreover, a cache manager is in charge of summary caching in memory and it can be bounded to a given memory requirement. Usually, less than a hundred of summaries are needed in the cache since this number covers the two or three top levels of even a wide hierarchy. Least recently used summaries are discarded when a required summary is not found in the cache.

On the other hand, the parallelization of the summary system is a key feature to ensure smooth scalability. As mentioned before, the implementation of the system is based on the Message-Oriented Programming paradigm. Each sub-system is autonomous and collaborates with the others through disconnected asynchronous method invocations. It is among the least demanding approaches in terms of availability and centralization. The autonomy of summary components allows for a distributed computing of the summary process. Once a component completes the treatment and evaluates the best operator for the hierarchy modification, if needed, a similar method is successively called on children nodes. The cache manager is able to handle several lists of summaries residing on different computers [15].

To summarize, our summary system combines advantages such as linear time complexity, controlled memory consumption, and a parallelized computing of the summarization process. Thanks to these advantages, we believe that this summary system is scalable in a distributed environment, and promises a successful integration in P2P systems.

2.3 Summary Representation

A summary z is a pair (I_z, R_z) where I_z is the intentional content of the summary and R_z is its extent, that is the group of database tuples described by I_z. The intent I_z provides a short description of z in terms of linguistic labels defined in the BK and used in the pre-processing step. Each descriptor is associated with two measures. First, a *Satisfaction degree* whose value reflects the accuracy of the descriptor regarding the ac-

tual content of the summary. Second, a *Support* that reflects the number of tuples within the summary that are actually represented by this descriptor.

For our purpose, we consider a summary as an indexing structure over distributed data in a P2P system. Thus, we add a third dimension to the definition of a summary z: a *peer-extent* P_z, which provides the set of peers having data described by z.

Definition 1 Peer-extent. *Let z be a node in a given hierarchy of summaries S, and P the set of all peers who participated to the construction of S. The* peer-extent *P_z of the summary z is the subset of peers owning, at least, one record of its extent R_z: $P_z = \{p \in P \mid R_z \cap R_p \neq \emptyset\}$, where R_p is the view over the database of node p, used to build summaries.*

Due to the above definition, we extend the notion of *data-oriented* summary in a given database, to a *source-oriented* summary in a given P2P network. In other words, our summary can be used as a database index (e.g. referring to relevant tuples), as well as a semantic index in a distributed system (e.g. referring to relevant nodes).

As it was mentioned before, a summary is an edge in the tree structure finally produced by the summarization service. The summary hierarchy S will be characterized by its *Coverage* in the P2P system; that is, the number of data sources described by S. Relative to the hierarchy S, we call *Partner Peer* a peer whose data is described by at least a summary node of S.

Definition 2 Partner peers. *The set of* Partner peers *P_S of a summary hierarchy S is the union of peer-extents of all the summary nodes: $P_S = \{\cup_{z \in S} P_z\}$.*

In the rest of this paper, we designate by "summary" a hierarchy of summaries maintained in a P2P system, unless otherwise specified.

3 Network Self-Organization

Intuitively, the term "self-organization" describes the ability of a P2P network to organize its participants into a cooperative framework, without the need of external intervention or control. For our purposes, we will understand *self-organization* as the capability of partitioning the network into domains, to optimally distribute data summaries, without using global information or restricting peer autonomy. In this section, we begin with an overview of the reasoning behind our solution, and then provide the algorithm for a network self-organization.

3.1 Rationale

A number of recent studies [12], [19] have shown that the existing networks have complex network characteristics, including power law degree distributions, small diameter, tolerance to node deletions, etc. Like the networks that it is designed for, our algorithm is completely decentralized, and mainly exploits the power-law link distribution in the node degree. Furthermore, in highly unstructured networks efficient algorithms should rely on local information in order to avoid a dependence on a central point of failure. In our solution, we suppose that a node knows only about the identities and the connectedness of its neighbors.

The key idea behind the solution is that random walks in power-law networks naturally gravitate toward the high degree nodes. A random walk is a technique proposed by [20] to replace flooding in unstructured P2P systems. At each step, a query message is forwarded to a randomly chosen neighbor until sufficient responses to the query are found. Although it makes better utilization of the P2P network than flooding, a random walk is essentially a blind search in that it does not take into account any indication of how likely it is the chosen node will have responses for the query. Adamic *et* al. [21] addressed this problem and showed that a better scaling is achieved by intentionally choosing high degree nodes. We will refer to this routing technique as "*selective walk*".

Our solution consists in identifying *summary peers* in a power law network with an exponent β, and maximum degree k_{max}[2]. Summary peers are defined as high-degree peers, which will serve as *centers of summary-attraction*. Using a selective walk which naturally and rapidly gravitates toward high degree nodes, a peer p finds the nearest summary peer SP to which it sends a duplicate of its local summary LS. The set of peers that discover the same summary peer SP are grouped around it and form a domain. These peers

[2] A distribution is said to be a power law distribution, if $P(k) \propto k^{-\beta}$, where β is called the exponent of the distribution

become *partners* relative to a global summary GS obtained by merging their local summaries. In [22], any node with degree k is considered as a high-degree node if $k \geq k_{max}/2$. However, k_{max} scales like $O(N^{1/\beta})$ [23] and is a global information. In the next section, we propose an IS_SUMPEER function that is executed locally at each peer to decide whether it is a summary peer or not, using minimum local information.

3.2 Algorithm

To our purpose, we have extracted a general model of a high-degree node in a power law network. Thus, the IS_SUMPEER function consists in matching this model with each node of the network. Algorithm 3.2 shows the steps involved in making this matching. First, we check if the current peer p is among the highest-degree peers in its neighborhood. In other words, the degree k of peer p should be greater than the median value of the set of its neighbor's degrees (i.e. $|subset\_inf| \succ |subset\_sup|$). Then, we verify if the local maximum degree max in p's neighborhood does not exceed $2 \cdot k$. Finally, we examine if k is larger than the mean value of neighbor's degrees by a constant ct. This condition makes a difference in the matching result when the neighbor's degrees follow an asymmetric distribution with a positive skew, i.e. there are a small number of very large degrees. In that case, the mean value is greater than the median. The constant ct permit to tune the selectivity of the matching function. The larger ct, less is the total number of summary peers.

Algorithm 1 $Is\_SumPeer$

1: **function** $Is\_SumPeer(k, NL)$
2: k is the degree of the current peer p, and NL is the Neighboring List that contains the identifiers of p's neighbors and their degrees.
3: $subset\_inf := p_i \; \forall \; 1 \leq i \leq |NL|$ such that $k_i \prec k$
4: $subset\_sup := p_i \; \forall \; 1 \leq i \leq |NL|$ such that $k_i \succ k$
5: $max :=$ **max** $(k_i), \forall \; 1 \leq i \leq |NL|$
6: $mean :=$ **mean** $(k_i), \forall \; 1 \leq i \leq |NL|$
7: **if** $(|subset\_inf| \succ |subset\_sup|)$ **and** $(k \succ max/2)$ **and** $(k \succ ct \cdot mean)$ **then**
8: $Is\_SumPeer :=$ true
9: **else** $Is\_SumPeer :=$ false
10: **end if**
11: **end function**

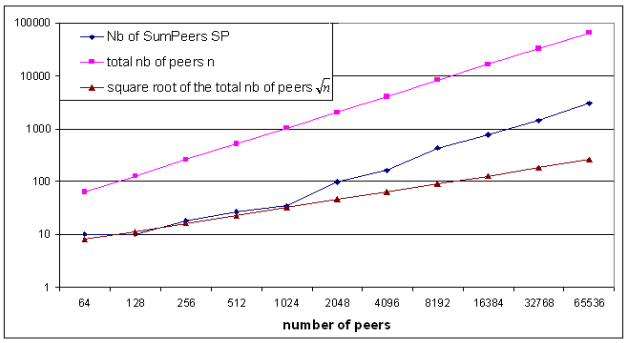

Figure 4. Number of summary peers vs. number of peers

Figure 4 shows the number of summary peers identified by our algorithm, in function of the total number of peers n. We see that this number is proportional to $\sqrt{n}$ for network sizes smaller than 1024, and is proportional to n for larger networks. Since our domains are formed around the summary peers, thus figure 4 gives directly the number d of domains obtained in the network. The shown results are similar to those found in [11]. It has been proved [11], theoretically and by simulation, that the optimal number of domains required to distribute a global index, is in $O(\sqrt{n})$ for total-lookup queries and in $O(n)$ for partial-lookup queries. A total-lookup query requires all results that are available in the system, whereas a partial-lookup query requires any m results, for some constant m. We believe that a total-lookup query is very difficult and costly in large P2P networks, and thus all queries are processed as being partial-lookup queries. Therefore, we conclude that using a local degree-based function, we can organize the network into an optimal number of domains: for total-lookup queries in small-sized networks, and for partial-lookup queries in larger-sized networks.

4 Summary management

In this section, we present our algorithms for summary construction and maintenance in a given domain. First, we work in a static context where all participants remain connected. Then we address the volatility of peers and propose appropriate solutions.

4.1 Summary Construction

We assume that each global summary is associated with a *Cooperation List* (CL) that provides information about its partner peers. An element of CL is composed of two fields. A partner identifier *PeerID*, and a 2-bit freshness value v that provides information about the description freshness as well as the availability of the corresponding database.

- value 0 (initial value): the descriptions are fresh relative to the original data,
- value 1: the descriptions need to be refreshed,
- value 2: the original data are not available. This value is used while addressing peer volatility in Section 4.3.

Algorithm 4.1 shows the messages exchanged between peers in order to build a global summary GS. A summary peer SP broadcasts a SUMPEER message that contains its identifier, to indicate its ability to host summaries. Since SP is supposed to have high connectivity, a small value of TTL (Time-To-Live) is sufficient to cover a large number of peers (e.g. $TTL = 2$). The message contains also a hop value h, initialized to 0, which is used to compute the distances between SP and the receiving peers.

A peer p who received a first SUMPEER message, maintains information about the corresponding summary peer SP (i.e. Line 16). Then, p sends to SP a LOCALSUM message that contains its local summary LS, and thus becomes a partner peer in the SP's domain. Upon receiving this last message, SP merges LS to its current global summary GS, and adds a new element in the cooperation list. However, a peer p who is already a partner may receive a new SUMPEER message. In that case, only if the new summary peer is nearer than the old one (based on latency), it chooses to drop its old partnership through a DROP message (i.e. Line 14), and it proceeds to participate to a new domain.

We now suppose that a peer p does not belong to any domain (is not a partner peer), and wants to participate to a global summary construction. Using a selective walk, it can rapidly find a summary peer SP (i.e. FIND message). The information about SP, which is maintained at each of its partners, makes

Algorithm 2 *Global Summary Construction*

```
1:  // Definition of different types of messages
2:  SUMPEER=⟨sender⟩ ⟨id, h, TTL⟩
3:  FIND=⟨sender⟩ ⟨h, TTL⟩
4:  LOCALSUM=⟨sender⟩ ⟨LS⟩
5:  DROP=⟨sender⟩ ⟨⟩
6:  // Treatment of messages
7:  Treat(msg)
8:  Switch msg.type
9:  // Receiving information about a summary peer
10: Case (SumPeer): msg.h++
11:     msg.TTL−−
12:     if (this.SumPeer=null) or (this.SumPeer.h≻msg.h)
    then
13:         if (this.IsPartner) then
14:             Send DROP message to this.SumPeer.id
15:         end if
16:         this.SumPeer := ⟨msg.id, msg.h⟩
17:         LOCALSUM := new msg (this.LS)
18:         Send LOCALSUM to msg.sender
19:         IsPartner := True
20:     end if
21:     if msg.TTL > 0 then
22:         Send msg to all neighbors
23:     end if
24: end Case
25: // Searching for a summary peer
26: Case (Find): msg.h++
27:     msg.TTL−−
28:     if (this.Is_SumPeer) then
29:         PEERSUM := new msg (this.id, msg.h, 1)
30:         Send PEERSUM to msg.sender
31:     else
32:         if (this.SumPeer ≠ null) then
33:             PEERSUM := new msg (this.SumPeer.id,
    (msg.h + this.SumPeer.h), 1)
34:             Send PEERSUM to msg.sender
35:         else
36:             if (msg.TTL > 0) then
37:                 p′ ← highest degree peer in N(p)
38:                 Send msg to p′
39:             end if
40:         end if
41:     end if
42: end Case
43: // arrival of a new partner
44: Case (LocalSum): CoopList.add (msg.sender, 0)
45:     GlobalSum := merge (GlobalSum, msg.LS)
46: end Case
47: // departure of a partner
48: Case (Drop): CoopList.remove (msg.sender)
49: end Case
```

the selective walk even shorter. Once a partner or a summary peer is reached, the FIND message is stopped (i.e. Line 32).

In the above algorithm, peers exchange summaries that are produced using local Background Knowledges (BKs). Thus, they may be represented in different user-defined vocabularies, making difficult their shared exploitation (e.g. merging operation). In this work, we assume that the participants to a collaborative database application agree on a *Common Background Knowledge* (CBK) that will be used locally by each summarization process. An example of such a CBK is the Systematized Nomenclature of Medicine Clinical Terms (SNOMED CT) [24], which is a comprehensive clinical terminology covering diseases, clinical findings, and procedures. On the other hand, several works have addressed the problem of semantic heterogeneity in advanced P2P applications (e.g. [25], [4]). Since our summaries are data structures that respect the original database schemas, we can assume that the techniques they proposed for a decentralized schema management can be also used to overcome the heterogeneity of summary representations, in the context of different BKs.

4.2 Summary Maintenance

A critical issue for any indexing structure is to maintain the index, relative to the current data instances, without incurring high costs. For a local summary, it has been demonstrated that the summarization process guarantees an incremental maintenance, using a *push* mode for exchanging data with the DBMS, while performing with a low complexity [15]. In this section, we propose a strategy for maintaining a global summary in a given domain, based on both *push* and *pull* techniques, in order to minimize the number of messages exchanged in the system. The appropriate algorithm is divided into two phases: data modification and summary reconciliation.

4.2.1 Push: Data Modification

Let GS be a global summary and P_{GS} the set of its partner peers. Each peer in P_{GS} is responsible for refreshing its own element in the GS's cooperation list. A partner peer p observes the modification rate issued on its local summary LS. When LS is considered as enough modified, the peer p sets its freshness value v to 1, through a *push message* to the corresponding summary peer SP. The value 1 indicates that the local summary version being merged while constructing GS does no longer correspond to the current instance of the database.

An important feature is that the frequency of push messages depends on modifications issued on local summaries, rather than on the underlying databases. It has been demonstrated in [15] that, after a given process time, a summary hierarchy becomes very stable. As more tuples are processed, the need to adapt the hierarchy decreases and hopefully, once all existing attribute combinations have been processed, incorporating new tuple consists only in sorting it in a tree. A summary modification can be detected by observing the appearance/disappearance of descriptors in summary intentions.

4.2.2 Pull: Summary Reconciliation

The summary peer SP, in its turn, observes the fraction of old descriptions (i.e. number of ones) in the cooperation list. Whenever this fraction exceeds a threshold value, the global summary GS must be refreshed. In that case, SP pulls all the partner peers to merge their current local summaries into the new version of GS, which will be then under reconstruction. The algorithm is described as follows.

SP initiates a reconciliation message that contains a new summary *NewGS* (initially empty). The message is propagated from a partner to another (started at SP). When a partner *p* receives this message, it first merges NewGS with its local summary. Then, it sends the message to another partner (chosen from the cooperation list CL). If *p* is the last visited peer, it sends the message to SP who will store the new version of the gobal summary. All the freshness values in CL are reset to zero. This strategy distributes the charge of summary merging on all partners, instead of imposing on SP to receive all local summaries and to make the merging calculations alone. Furthermore, this strategy guarantees a high availability of the global summary, since only one update operation is performed at the end by SP.

4.3 Peer Dynamicity

In large P2P systems, a peer connects mainly to download some data and may leave the system without any constraint. Therefore, the shared data can be submitted with a low modification rate, while the rate of node arrival/departure is very important. We now study the effect of this peer dynamicity on our summary management algorithms, and propose appropriate solution.

4.3.1 Partner Peer Arrival/Departure

In unstructured P2P systems, when a new peer p joins the system, it contacts some existing peers to determine the set of its neighbors. If one of these neighbors is a partner peer, p sends its local summary LS to the corresponding summary peer SP, and thus becomes a new partner in the SP's domain. SP adds a new element to the cooperation list with a freshness value v equal to one. Recall that the value 1 indicates the need of pulling peer p to get new data descriptions.

When a partner peer p decides to leave the system, it first sets its freshness value v to two in the cooperation list, through a push message. This value reminds the participation of the disconnected peer p to the corresponding global summary, but also indicates the unavailability of the original data. There are two alternatives to deal with such a freshness value. First, we can keep the data descriptions and use it, when a query is approximately answered using the global summary. A second alternative consists in considering the data descriptions as expired, since the original data are not accessible. Thus, a partner departure will accelerate the summary reconciliation initiating. In the rest of this paper, we adopt the second alternative and consider only a *1*-bit freshness value v: a value 0 to indicate the freshness of data descriptions, and a value 1 to indicate either their expiration or their unavailability.

However, if peer p failed, it could not notify its summary peer by its departure. In that case, its data descriptions will remain in the global summary until a new summary reconciliation is executed. The reconciliation algorithm does not require the participation of a disconnected peer. The global summary GS is reconstructed, and descriptions of unavailable data will be then omitted.

4.3.2 Summary Peer Arrival/Departure

In section 3, we have presented our IS_SUMPEER function that is executed at each peer to decide whether it is a summary peer or not. This function is based on node connectivity, and thus variations of node degrees may incur modifications in the function results. Therefore, we suppose that a peer executes periodically the IS_SUMPEER function.

However, we believe that the results of the IS_SUMPEER function do not change frequently thanks to two characteristics of P2P networks. First, connections tend to be formed preferentially because peers tend to discover high-degree nodes in the network overlay [12]. Second, although the nodes join and leave the network with a high rate, we suppose that each node leave is very probably accompanied by a new node join such the the total number of nodes remains the same. Thus, the cases in which a summary peer becomes an ordinary peer, or a node is submitted to a significant degree variation rarely occur. However, when a new highly-available peer attracts many peer connections and becomes a summary peer, it simply diffuses this information as described in section 4.1, and a new domain starts to appear around it.

Now, when a summary peer SP decides to leave the system, it sends a release message to all its partners using the cooperation list. Upon receiving such a message, a partner p makes a selective walk to find a new summary peer. However, if SP failed, it could not notify its partners. A partner p who has tried to send push or query messages to SP will detect its departure and thus search for a new one.

4.4 Capacity-based Summary Distribution

So far, we have considered a centralized approach for storing a global summary in a given domain. Each summary peer SP is in charge of storing the global summary GS of its domain. However, this implies that each query posed by a partner p will be sent to SP, which may be then overloaded. Here we give a simple solution to distribute GS, and thus the query load, based on node capacity. In [], capacities are assigned to nodes based on a distribution that is derived from the measured bandwidth distributions for Gnutella, as

Capacity level	percentage of nodes
$1x$	20%
$10x$	45%
$100x$	30%
$1000x$	4.9%
$10000x$	0.1%

Table 1. Gnutella-like node capacity distributions

reported by Saroiu *et al* []. The capacity distribution has five levels of capacity, each separated by an order of magnitude (Table 1). A node i is modeled as possessing a capacity C_i, which represents the number of queries that it can process per unit time. In addition to its capacity, each node i is assigned a query generation rate q_i, which is the number of queries that it generates per unit time. Therefore, a node i is represented by: $i = (C_i, q_i)$.

Now, we suppose that each summary peer SP delegates an *Assistant Peer* AP, which is the highest-capacity neighbor that belongs to the same domain. The summary peer SP accepts n_1 peers as partners among the n_d peers of its domain d such that: $\sum_{i=1}^{n_1} q_i \prec C_{SP}$. If n_1 is greater than n_d, SP is considered as a high-capacity peer, and thus will host the global summary of the entire domain. Otherwise, we adopt a *deviation* method to distribute the charge between the summary peer and its assistant. When SP receives a LOCALSUM message (see algorithm 4.1) and it cannot support the query generation rate of the sender, it deviates the message to the assistant AP. In this case, the summary peer SP serves as a guide. Using a selective walk, a peer p rapidly finds SP that sends him to the assistant AP, in order to handle a duplicate of its local summary. However, the global summary GS of the domain d is divided into two summaries: GS_1 and GS_2, maintained by SP and AP respectively. Thus, to allow each partner peer exploiting both summaries, we replicate GS_2 at SP and GS_1 at AP, as shown by Figure 5.

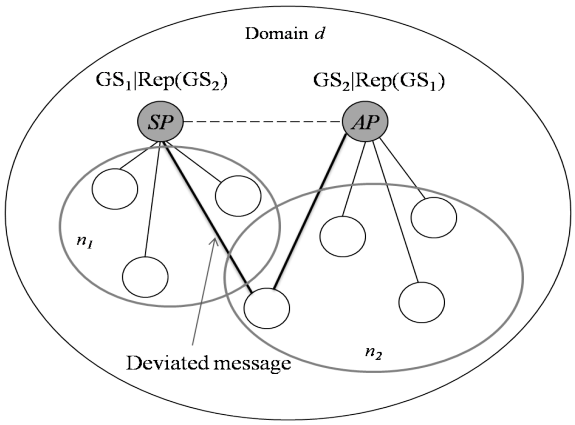

Figure 5. capacity-based summary distribution

5 Query Processing

In this section, we describe how a query Q, posed at a peer p, is processed. For query routing, we adopt a hybrid approach: intra-domain summary querying, and inter-domain query flooding. For query answering, our mechanism is able to achieve two distinct tasks depending on the user/application requirements: *peer localization* to return the original result records, and *summary answering* to return approximate answers.

Peer p first sends the query Q to the summary peer SP of its domain. SP proceeds then to query the available global summary GS, which is considered as a semantic index over the data shared in the domain. This allows efficient peer localization since we exploit data descriptions rather than structural information on database schemas (e.g. [25]). Summary querying is divided into two phases: 1) query extension and 2) query evaluation.

5.1 Query Extension

First, the query Q must be extended to a flexible query Q^* in order to be handled by the summary querying process. For instance, consider the following selection query Q:
SELECT BMI FROM PATIENT WHERE $age < 30$ AND DISEASE = "MALARIA"

This phase consists in replacing the original value of each selection predicate by the corresponding descriptors defined in the Background Knowledge (*BK*). According to the fuzzy partition of Figure 3, the above query is transformed to Q^*:

SELECT BMI FROM PATIENT WHERE AGE IN {YOUNG, ADULT} AND DISEASE = "MALARIA"

Let QS (resp.QS^*) be the *Query Scope* of query Q (resp.Q^*) in the domain, that is, the set of peers that should be visited to answer the query. Obviously, the query extension phase may induce false positives in query results. To illustrate, a 35 years old patient will be returned as an answer to the query Q^*, while the selection predicate on the attribute `age` of the original query Q is not satisfied. However, false negatives cannot occur, which is expressed by the following inclusion: $QS \subseteq QS^*$.

In the rest of this paper, we suppose that a user query is directly formulated using descriptors defined in the BK (i.e. $Q = Q^*$). As we discussed in the introduction of this work, a doctor that participates to a given medical collaboration, may ask query Q like "the BMI of *young* and *adult* patients diagnosed with *malaria*". Thus, we eliminate potential false positives that may result from query extension.

5.2 Query Evaluation

This phase deals with matching a set of summaries organized in a hierarchy S, against the query Q. The query is transformed into a logical proposition P used to qualify the link between a summary node and the query. Proposition P is under a conjunctive form in which all descriptors appears as literals. In consequence, each set of descriptors yields on corresponding clause. For instance, the above query Q is transformed to $P =$ *(young OR adult) AND (malaria)*.

A valuation function has been defined to valuate the proposition P in the context of a summary node z. Then, a selection algorithm performs a fast exploration of the hierarchy and returns the set Z_Q of most precise summaries that satisfy the query. For more details see [26]. Once Z_Q determined, the query evaluation process is able to achieve two distinct tasks: 1) Peer localization, and 2) Summary answering.

5.2.1 Peer Localization

Since the extended definition of a summary node z provides a peer-extent, i.e. the set of peers P_z having data described by its intent (see Definition 1), we can define the set P_Q of relevant peers for the query Q as follows: $P_Q = \{\cup_{z \in Z_Q} P_z\}$. The query Q is directly propagated to these relevant peers. However, the efficiency of this query routing depends on the completeness and the freshness of summaries, since stale answers may occur in query results. We define a *False Positive* as the case in which a peer p belongs to P_Q and there is actually no data in the p source that satisfies Q (i.e. $p \notin QS$). A *False Negative* is the reverse case in which a p does not belong to P_Q, whereas there exists at least one tuple in the p data source that satisfies Q (i.e. $p \in QS$).

5.2.2 Summary Answering

A distinctive feature of our approach is that a query can be processed entirely in the summary domain. An approximate answer can be provided from summary descriptions, without having to access original, distributed database records. The selected summaries Z_Q are aggregated according to their interpretation of proposition P: summaries that have the same required characteristics on all predicates (i.e. *age* and *disease*) form a class. The aggregation in a given class is a union of descriptors: for each attribute of the selection list (i.e. *BMI*), the querying process supplies a set of descriptors which characterize summaries that respond to the query through the same interpretation [26]. For example, for the class $\{young, malaria\}$, we can obtain an output set *BMI* = $\{underweight, normal\}$.

Now, we suppose that processing a query Q in a given domain d_i returns C_i results, while the user requires C_t results. We note that, if C_t is less than the total number of results available in the network, Q is said to be a *partial-lookup* query. Otherwise, it is a *total-lookup* query. Obviously, when C_i is less than C_t, the query should be propagated to other domains. To this end, we adopt the following variation of the flooding mechanism.

Let P_i the subset of peers that have answered the query Q in the domain d_i: $|P_i| = (1 - FP) \cdot |P_Q|$, where FP is the fraction of false positives in query results. The query hit in the domain is given by: $(|P_i| \,/\, |d_i|)$. As shown by many studies, the existing P2P networks have small-world features [27]. In such a context, users tend to work in groups. A group of users, although not always located in geographical proximity, tends to use the same set of resources

(i.e. *group locality* property). Thus, we assume that the probability of finding answers to query Q in the neighborhood of a relevant peer in P_i, is very high since results are supposed to be *nearby*. This probability is also high in the neighborhood of the originator peer p since some of its neighbors may be interested in the same data, and thus have cached answers to similar queries. Therefore, the summary peer SP_i of domain d_i sends a flooding request to each peer in P_i as well as to peer p. Upon receiving this request, each of those peers broadcasts the query with a limited value of TTL. Once a new domain is reached or TTL becomes zero, the query is stopped.

Besides, the summary peer SP which is a high-degree peer may have some *long-range* links, that is, links to other domains. Thus, SP also sends the flooding request to the subset of its neighbors that do not belong to its current domain (neighbors that do not figure in its cooperation list). This will accelerate covering a large number of domains. In each visited domain, the query is processed as described above. When the number of query results becomes sufficient (i.e. larger than C_t), or the network is entirely covered, the query routing is terminated.

6 Performance evaluation

In this section, we devise a simple model of the summary management cost. Then, we evaluate and analyze our model with a simulation using the BRITE topology generator and SimJava.

6.1 Cost Model

A critical issue in summary management is to trade off the summary updating cost against the benefits obtained for queries.

6.1.1 Summary Update Cost

Here, our first undertaking is to optimize the update cost while taking into account *query accuracy*. In the next section, we discuss query accuracy which is measured in terms of the percentage of false positives and false negatives in query results. The cost of updating summaries is divided into: usage of peer resources, i.e. time cost and storage cost, and the traffic overhead generated in the network.

Time Cost A unique feature of SAINTETIQ is that the changes in the database are reflected through an incremental maintenance of the summary hierarchy. The time complexity of the summarization process is in $O(n)$ where n is the number of tuples to be incorporated in that hierarchy [15].

For a global summary update, we are concerned with the complexity of merging summaries. The MERGING method that has been proposed is based on the SAINTETIQ engine. This method consists in incorporating the leaves of a given summary hierarchy S_1 into an another S_2, using the same algorithm described by the SAINTETIQ summarization service (referenced in Section 2.2.3). It has been proved that the complexity C_{M12} of the MERGING(S_1, S_2) process is constant w.r.t the number of tuples. More precisely, C_{M12} depends on the maximum number of leaves of S_1 to incorporate into S_2. However, the number of leaves in a summary hierarchy is not an issue because it can be adjusted by the user according to the desired precision. A detailed Background Knowledge (BK) will lead to a greater precision in summary description, with the natural consequence of a larger summary. Moreover, the hierarchy is constructed in a top-down approach and it is possible to set the summarization process so that the leaves have any desired precision.

Storage Cost We denote by k the average size of a summary node. In the average-case assumption, there are $\sum_{i=0}^{d} B^i = (B^{d+1} - 1)/(B - 1)$ nodes in a B-arity tree with d, the average depth of the hierarchy. Thus the average space requirement is given by: $C_m = k.(B^{d+1} - 1)/(B - 1)$. Based on real tests, $k = 512$ bytes gives a rough estimation of the space required for each summary node. An important issue is that the size of the hierarchy is quite related to its stabilization (i.e. B and d). As more tuples are processed, the need to adapt the hierarchy decreases and incorporating a new candidate tuple may consist only in sorting a tree. Hence, the structure of the hierarchy remains stable and no additional space is required. On the other hand, when we merge two hierarchies S_1 and S_2 having sizes of C_{m1} and C_{m2} respectively, the size of the resultant hierarchy is always in the order of the $max\,(C_{m1}, C_{m2})$. However, the size of a summary hierarchy is limited to a maximum value which corre-

sponds to a maximum number of leaves that cover all the possible combinations of the BK descriptors. Thus, storing the global summary at the summary peer is not a strength constraint.

According to the above discussion, the usage of peer resources is optimized by the summarization process itself, and the distribution of summary merging while updating a global summary. Thus, we restrict now our focus to the traffic overhead generated in the P2P network.

Network Traffic Recall that there are two types of exchanged messages: *push* and *reconciliation*. Let local summaries have an average lifetime of L seconds in a given global summary. Once L expired, the node sends a (push) message to update its freshness value v in the cooperation list CL. The reconciliation algorithm is then initiated whenever the following condition is satisfied: $\sum_{v \in CL} v/|CL| \geq \alpha$, where α is a threshold that represents the ratio of old descriptions tolerated in the global summary. During reconciliation, only one message is propagated among all partner peers until the new global summary version is stored at the summary peer SP. Let F_{rec} be the reconciliation frequency. The update cost is:

$$C_{up} = 1/L + F_{rec} \ \ messages\ per\ node\ per\ second \quad (1)$$

In this expression, $1/L$ represents the number of push messages which depends either on the modification rate issued on local summaries or the connection/disconnection rate of peers in the system. Higher is the rate, lower is the lifetime L, and thus a large number of push messages are entailed in the system. F_{rec} represents the number of reconciliation messages which depends on the value of α. This threshold is our system parameter that provides a trade-off between the cost of summary updating and query accuracy. If α is large, the update cost is low since a low frequency of reconciliation is required, but query results may be less accurate due both to false positives stemming from the descriptions of non existent data, and to false negatives due to the loss of relevant data descriptions whereas they are available in the system. If α is small, the update cost is high but there are few query results that refer to data no longer in the system, and nearly all available results are returned by the query.

6.1.2 Query Cost

When a query Q is posed at a peer p, it is first matched against the global summary available at the summary peer SP of its domain, to determine the set of relevant peers P_Q. Then, Q is directly propagated to those peers. The query cost in a domain d is given by:

$$C_d = (1 + |P_Q| + (1 - FP) \cdot |P_Q|) \ \ messages,$$

where $(1 - FP) \cdot |P_Q|$ represents the query responses messages (i.e. query hit in the domain).

Here we note that, the cooperation list CL associated with a global summary provides information about the relevance of each database description. Thus, it gives more flexibility in tuning the *recall/precision* trade-off of the query answers in domain d. The set of all partner peers P_H in CL can be divided into two subsets: $P_{old} = \{p \in P_H \mid p.v = 1\}$, the set of peers whose descriptions are considered old, and $P_{fresh} = \{p \in P_H \mid p.v = 0\}$ the set of peers whose descriptions are considered fresh according to their current data instances. Thus, if a query Q is propagated only to the set $V = P_Q \cap P_{fresh}$, then precision is maximum since all visited peers are certainly matching peers (no false positives), but recall depends on the fraction of false negatives in query results that could be returned by the set of excluded peers $P_Q \backslash P_{fresh}$. On the contrary, if the query Q is propagated to the extended set $V = P_Q \cup P_{old}$, the recall value is maximum since all matching peers are visited (no false negatives), but precision depends on the fraction of false positives in query results that are returned by the set of peers P_{old}.

Now we consider that the selectivity of query Q is very high, such that each relevant peer has only one result tuple. Thus, when a user requires C_t tuples, we have to visit C_t relevant peers. The cost of inter-domain query flooding is given by:

$$C_f = ((1 - FP) \cdot |P_Q| + 2) \cdot \sum_{i=1}^{TTL} k^i \ \ messages,$$

where k is the average degree value in an unstructured P2P system (e.g. average degree of 3.5, similar to Gnutella-type graphs). Remember that, the set of relevant peers who have answered the query (i.e. $(1-FP) \cdot |P_Q|$), the originator and the summary peers participate to query flooding. In this expression, we

Parameter	value
Network configuration	
local summary lifetime L	skewed distribution, Mean=3h, Median=1h
number of peers n	16–5000
Workload configuration	
number of queries q	200
matching nodes/query *hits*	10%
System parameter	
freshness threshold α	0.3–0.8

Table 2. Simulation Parameters

consider that a summary peer has on average k long-range links. As a consequence, the total cost of a query is:

$$C_Q = C_d \cdot \frac{C_t}{(1-FP)\cdot|P_Q|} + C_f \cdot (1 - \frac{C_t}{(1-FP)\cdot|P_Q|}) \quad (2)$$

In this expression, the term $C_t/((1-FP)\cdot|P_Q|)$ represents the number of domains that should be visited. For example, when $C_t = ((1-FP)\cdot|P_Q|)$, one domain is sufficient and no query flooding is required.

6.2 Simulation

We evaluated the performance of our solutions through simulation, based on the above cost model. First, we describe the simulation setup. Then we present simulation results to evaluate various performance dimensions and parameters: scale up, query accuracy, effect of the freshness threshold α.

6.2.1 Simulation Setup

For our simulation, we used the SimJava package [28] and the BRITE universal topology generator [29]. We simulate a n-node P2P system, assigning a data source with t tuples to each node. We calibrate our simulator using real data gathered by Saroiu et al. [12], in a study of the Gnutella file-sharing network. The simulation parameters are shown in Table 2 and fall into three categories: network parameters, workload parameters, and system parameters.

In our tests, we consider that local summary lifetimes are quite related to the node lifetimes, since the rate of node connection/disconnection is supposed to

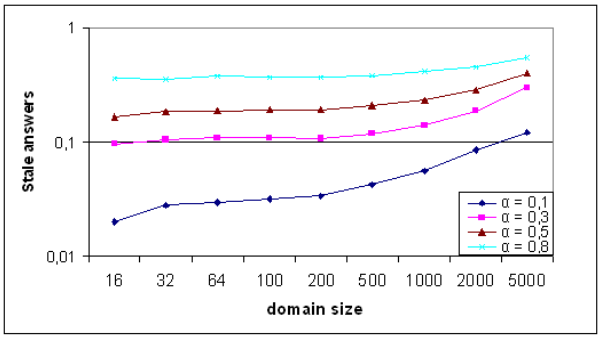

Figure 6. Stale answers vs. domain size

be greater than the modification rate issued on local summaries, and this for two reasons. First, in large P2P systems, we mainly deal with selection queries to locate and download required data. Thus, the original data are submitted to a low modification rate. Second, our summaries are even more stable than the original data (as we discussed before). Thus, the volatility of peers is, in reality, the main reason for a global summary reconciliation. Under this assumption, we consider that local summary lifetimes, like node lifetimes, follow a skewed distribution with a mean lifetime of 3 hours, and a median lifetime of 60 minutes.

Our workload has 200 queries. The query rate is 0.00083 queries per node per second (one query per node per 20 minutes) as suggested in [30]. Each query is matched by 10% of the total number of peers. Finally, Our system parameter α that decides of the reconciliation frequency varies between 0.1 and 0.8.

6.2.2 Update Cost

In this set of experiments, we quantify the trade-off between query accuracy and the cost of updating a global summary in a given domain. Figure 6 depicts the fraction of stale answers in query results for different values of the threshold α. Here, we illustrate the worst case. For each partner peer p having a freshness value equal to 1, if it is selected in the set P_Q then it is considered as false positive. Otherwise, it is considered as false negative. However, this is not the real case. Though it has a freshness value equal to 1, the peer p does not incur stale answers unless its database is changed relative to the posed query Q. Thus, Figure 6 shows the worst, but very reasonable values. For instance, the fraction of stale answers is limited to 11%

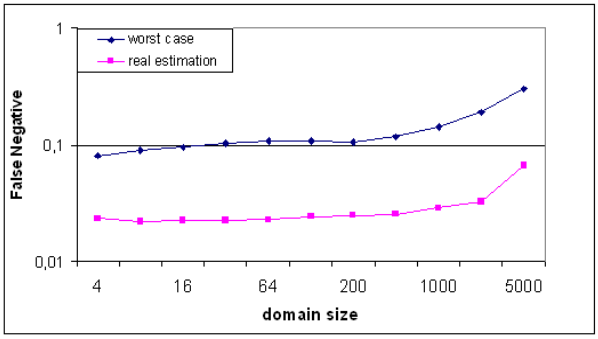

Figure 7. False negative vs. domain size

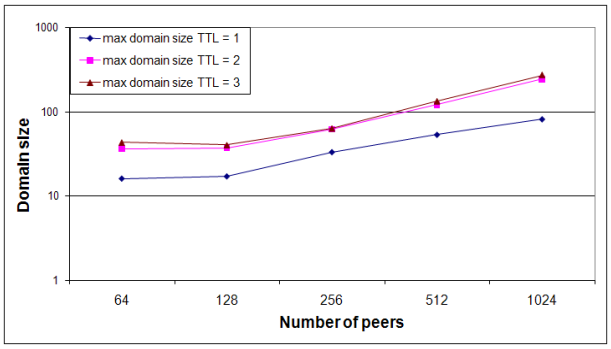

Figure 8. Max domain size vs. network size

for a network of 500 peers when the threshold α is set to 0.3 (30% of the peers are tolerated to have old/non existent descriptions).

Moreover, Figure 8 shows the maximum domain sizes obtained in our self-organized network, for different TTL values. Recall that the TTL value is used to broadcast a SUMPEER message (see Algorithm 4.1). The maximum domain size is approximatively less than 25% of the total number of peers. Thus, in Figure 6, the fraction of stale answers measured for a domain size of 256 peers, corresponds to a network of size 1024.

As mentioned in Section 6.1.2, if we choose to propagate the query only to the set $V = P_Q \cap P_{fresh}$ we eliminate the possible false positives in query results. However, this may lead to additional false negatives. Figure 7 shows the fraction of false negatives in function of the domain size. Here we take into account the probability of the database modification relative to the query, for a peer having a freshness value equal to 1. We see that the fraction of false negatives is limited to 3% for a domain size less than 2000 (i.e. network

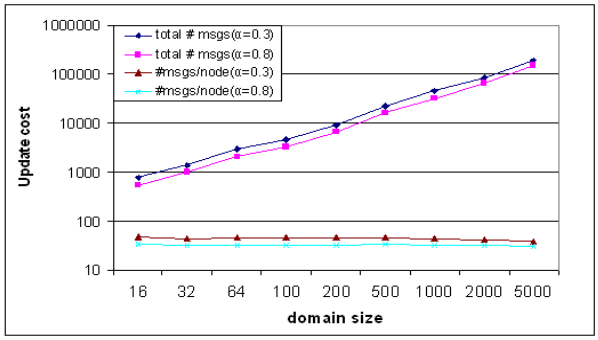

Figure 9. number of messages vs. domain size

size less than 8000). The real estimation of stale answers shows a reduction by a factor of 4.5 with respect to the preceded values.

Figure 9 depicts the update cost in function of the domain size, and this for two threshold values. The total number of messages increases with the domain size, but not surprisingly, the number of messages per node remains almost the same. In the update cost equation 6.1.1, the number of push messages for a given peer is independent of domain size. On the other hand, the number of reconciliation messages decreases slowly with the number of peers, for a given value of the threshold α. More interestingly, when the threshold value decreases (from 0.8 to 0.3) we notice a little cost increasing of 1.2 on average. For a domain of 1000 peers, the update cost increases from 0.01056 to 0.01296 messages per node per minute (not shown in figure). However, a small value of the threshold α allows to reduce significantly the fraction of stale answers in query results, as seen in Figure 6. We conclude therefore that tuning our system parameter, i.e. the threshold α, do not incur additional traffic overhead, while improving query accuracy.

6.2.3 Query Cost

In this set of experiments, we compare our algorithm for query processing against centralized-index and pure non-index/flooding algorithms. A centralized-index approach is very efficient since a single message allows locating relevant data. However, a central index is vulnerable to attack and it is difficult to keep it up-to-date. Flooding algorithms are very used

in real life, due to their simplicity and the lack of complex state information at each peer. A pure flooding algorithm consists in broadcasting the query in the network till a stop condition is satisfied, which may lead to a very high query execution cost. Here, we limit the flooding by a value 3 of TTL (Time-To-Live).

According to Table 2, the query hit is 10% of the total number of peers. For our query processing approach, which is mainly based on summary querying (SQ), we consider that each visited domain provides 10% of the number of relevant peers (i.e. 1% of the network size). In other words, we should visit 10 domains for each query Q. From equation 6.1.2, we obtain: $C_Q = (10 \cdot C_d + 9 \cdot C_f)$ messages. Figure 10 depicts the number of exchanged messages to process a query Q, in function of the total number of peers. The centralized-index algorithm shows the best results that can be expected from any query processing algorithm, when the index is complete and consistent, i.e. the index covers the totality of data available in the system, and there are no stale answers in query results. In that case, the query cost is: $C_Q = 1 + 2 \cdot ((0.1) \cdot n)$ messages, which includes the query message sent to the index, the query messages sent to the relevant peers and the query response messages returned to the originator peer p.

In Figure 10, we observe that our algorithm SQ shows good results by significantly reducing the number of exchanged messages, in comparison with a pure query flooding algorithm. For instance, the query cost is reduced by a factor of 3.5 for a network of 2000 peers, and this reduction becomes more important with a larger-sized network. We note that in our tests, we have considered the worst case of our algorithm, in which the fraction of stale answers of Figure 6 occurs in query results (for $\alpha = 0.3$).

7 Comparison with Related Work

Traditionally, there have been two categories of P2P search systems: Gnutella-style unstructured systems [8] and structured systems. The first relies on flooding mechanism and its variations to propagate a query. Though simple and robust, this approach suffers from high query execution and poor query recall. Distributed Hash Tables have been proposed for a variety of distributed applications. Systems such as Pastry [10], Chord [1] and CAN [2] build a distributed hash table on the top of the overlay to organize data in the network. Although these systems offer an efficient search, they compromise peer autonomy. Furthermore, data is located using unique and globally known data identifiers; complex queries are difficult to support.

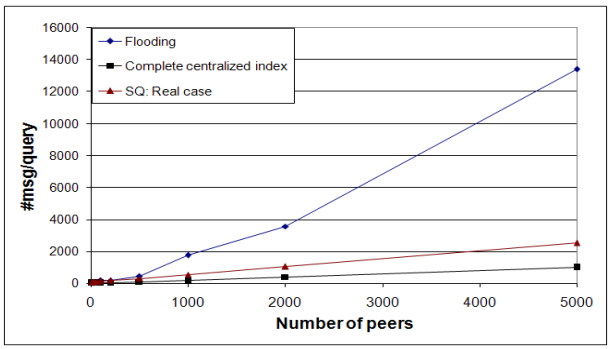

Figure 10. Query cost vs. number of peers

Other works have focused on schema mediation in P2P systems. In Edutella [31], each peer stores locally data described in RDF relatively to some existing reference ontologies. However, the system requires a strict topology with hypercubes and the use of super peers. Piazza [4] offers a more decentralized approach for schema management. A query is propagated along pre-existing pairwise mappings between peer schemas. However, when the number of peer increases, the number of possible mapping paths increases and thus, not all matching peers are visited. Furthermore, longer mapping paths require to reformulate a query many times which lead to semantic loss.

So far, research on P2P systems has focused on providing distributed data management techniques (e.g. query processing, schema management, data replication, etc.) that can scale with the number of participants in the network. However, an emergent issue is that new advanced P2P applications generate huge amount of data, which requires other data management techniques such as "data summarization". Downsizing massive data sets allows to address some critical issues such as individual data obfuscation, optimization of the usage of system resources like storage space and network bandwidth, as well as effective approximate answers to queries.

However, the work on database summarization has been done in a centralized environment. The approaches that have been proposed in the literature can be classified into three categories. The first one focuses on aggregate computation (e.g. [32, 33]). A second class of approaches extends the previous one in that it tries to produce more compact representations of aggregates (e.g. [13, 34]). the main challenge for such methods is to keep expressiveness of the provided access methods (aggregate queries) to the items without any need to uncompress the structure. The third family of approaches (e.g. [35, 36]) deals with intentional characterization of groups of individuals based on usual mining algorithms. This work consists in introducing database summaries in P2P networks, to address their scalability in terms of both amount of shared data and number of participants. Our approach for database summarization is based on SAINTETIQ [13] that uses the fuzzy set theory [16] to build robust summaries, using linguistic variables [14].

8 Conclusion

In this paper, we proposed a model for summary management in unstructured P2P systems. The innovation of this proposal consists in combining the P2P and database summarization paradigms, in order to support data sharing on a world wide scale. The database summarization approach that we proposed provides efficient techniques for data localization as well as for data description in P2P systems. In fact, our summaries are semantic indexes that support locating relevant data based on their content. Besides, an important feature is that these summaries are compact data descriptions that can approximately answer a query without retrieving original records from huge, highly distributed databases.

We made two main contributions. First, we defined a new function for organizing the network into domains, in order to distribute summaries built over the shared data. This function is completely decentralized and uses only local information, which avoids a dependence on a single point of failure. Our simulation results showed that this function allows partitioning the network into an optimal number of domains: for total-lookup queries in small-sized networks, and for partial-lookup queries in larger-sized networks. Second, we proposed efficient algorithms for summary management in a given domain. Our performance evaluation showed that the cost of query routing in the context of summaries is significantly reduced in comparison with flooding algorithms, without incurring high costs of summary maintenance.

References

[1] I.Stoica, R.Morris, D.Karger, M.F.Kaashoek, H.Balakrishnan: Chord: A scalabale peer-to-peer lookup service for internet applications. In: Proc SIGCOMM. (2001)

[2] S.Ratnasamy, P.Francis, M.Handley, R.M.Karp, S.Shenker: A scalable content–addressable network. In: Proc SIGCOMM. (2001)

[3] A.Crespo, H.G.Molina: Routing indices for peer-to-peer systems. In: Proc Conference on Distributed Computing Systems. (2002)

[4] I.Tartinov, *et* al: The Piazza peer data management project. In: Proc SIGMOD. (2003)

[5] A.Crespo, H.G.Molina: Semantic overlay networks for p2p systems. Technical report, Computer Science Department, Stanford University (2002)

[6] A.Oser, F.Naumann, W.Siberski, W.Nejdl, U.Thaden: Semantic overlay clusters within super-peer networks. In: Proc of the International Workshop on Databases, Information Systems and Peer-to-Peer Computing in Conjunction with the VLDB. (2003)

[7] G.Koloniari, Y.Petrakis, E.Pitoura: Content–based overlay networks of XML peers based on multi-level bloom filters. In: Proc VLDB. (2003)

[8] http://www.gnutella.com

[9] http://www.napster.com

[10] A.Rowstron, P.Druschel: Storage management and caching in PAST, a large–scale, persistent peer-to-peer storage utility. In: Proc Symposium on Operating Systems Principles(SOSP). (2001)

[11] P.Ganesan, Q.Sun, H.G.Molina: Adlib: a self-tuning index for dynamic peer-to-peer systems. In: Proc ICDE. (2005)

[12] S.Saroiu, P.Gummadi, S.Gribble: A measurement study of peer-to-peer file sharing systems. In: Proc of Multimedia Computing and Networking (MMCN). (2002)

[13] G.Raschia, N.Mouaddib: A fuzzy set-based approach to database summarization. Fuzzy sets and systems 129(2) (2002) 137–162

[14] L.A.Zadeh: Concept of a linguistic variable and its application to approximate reasoning-I. Information Systems **8** (1975) 199–249

[15] R.Saint-Paul, G.Raschia, N.Mouaddib: General purpose database summarization. In: Proc VLDB. (2005)

[16] L.A.Zadeh: Fuzzy sets. Information and Control **8** (1965) 338–353

[17] L.A.Zadeh: Fuzzy sets as a basis for a theory of possibility. Fuzzy Sets and Systems **100** (1999) 9–34

[18] K.Thompson, P.Langley: Concept formation in structured domains. In: Concept formation: Knowledge and experience in unsupervised learning. Morgan Kaufmann, CA 127–161

[19] M.Ripeanu, I.Foster, A.Iamnitchi: Mapping the gnutella network: Properties of large-scale peer-to-peer systems and implications for system design. IEEE Internet Computing Journal **6**(1) (2002)

[20] Q.Lv, *et* al: Search and replication in unstructured peer-to-peer networks. In: Proc Int.conference on Supercomputing. (2005)

[21] L.Adamic, *et* al: Search in power law networks. Physical Review E **64** (2001) 46135–46143

[22] N.Sarshar, P.Boykin, V.Roychowdhury: Percolation search in power law networks: Making unstructured peer-to-peer networks scalable. P2P (2004) 2–9

[23] W.Aiello, F.Chung, L.Lu: A random graph model for massive graphs. In: Proc symposium on Theory of computing (STOC). (2000)

[24] : http://www.snomed.org/snomedct

[25] R.Akbarinia, V.Martins, E.Pacitti, P.Valduriez: Design and implementation of APPA. In: Global Data Management (Eds. R. Baldoni, G. Cortese and F. Davide). (2006)

[26] W.A.Voglozin, G.Raschia, L.Ughetto, N.Mouaddib: Querying the SAINTETIQ summaries–a first attempt. In: Proc Int.Conf.On Flexible Query Answering Systems (FQAS). (2004)

[27] A.Iamnitchi, M.Ripeanu, I.Foster: Locating data in (small-world?) peer-to-peer scientific collaborations. In: Proc IPTPS: Revised Papers from the First International Workshop on Peer-to-Peer Systems. (2002)

[28] F.Howell, R.McNab: Simjava: a discrete event simulation package for java with the applications in computer systems modeling. In: Proc Int. Conf on Web-based Modelling and Simulation, Society for Computer Simulation. (1998)

[29] http://www.cs.bu.edu/brite/

[30] B.Yang, H.G.Molina: Comparing hybrid peer-to-peer systems. In: Proc VLDB. (2001)

[31] W.Nedjl, *et* al: Edutella: a p2p networking infrastructure based on rdf. In: WWW02. (2002)

[32] A.Shoshani: Statistical databases: Characteristics, problems, and some solutions. In: Proc VLDB, Morgan Kaufmann (1982)

[33] A.Shoshani: OLAP and statistical databases: Similarities and differences. In: Proc SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems. (1997)

[34] L.Lakshmanan, J.Pei, J.Han: Quotient cube: How to summarize the semantics of a data cube. In: Proc VLDB. (2002)

[35] H.Jagadish, R.Ng, B.Ooi, A.Tung: Itcompress: An iterative semantic compression algorithm. In: Proc ICDE. (2004)

[36] S.Babu, M.Garofalakis, R.Rastogi: Spartan: A model-based semantic compression system for massive data tables. In: Proc SIGMOD. (2001)